MILES BERRY PRINCIPAL LECTURER

ONE LANGUAGE AT A TIME

Do students need breadth of experience or depth of skills?

Most of us working in computer science education seem to agree that coding is not the end in itself, but the means through which our students get useful things done, express themselves creatively, and develop their understanding of the fundamental principles of computer science. While some of our students will go on to be the software engineers and computer scientists of the future, almost all will find themselves using computers. An understanding of how these machines work, and how to think about problems so that the computers can help solve them, is what's important for these students.

> in short, they need to become fluent (or at least conversant) in one, or two, or three programming languages

One language or many?

Where some of us disagree is on the best means to this end. On the one hand, there are those who argue that these principles, and indeed the constructs of programming (such as sequence, selection, and repetition), are best learnt generically and then applied in as many programming languages as time, teacher expertise, and student interest permits. Others take the view that actual, practical experience of coding in one language is the best way to learn to program – and through this to acquire, either through explicit instruction or guided discovery, an understanding of the underlying principles.

The arguments for the 'many languages' approach seem similar to those used in the ICT teaching past: that to avoid death by PowerPoint, we should allow students to choose Prezi, Google Slides etc. instead, as the software skills themselves are what matter. The 'many languages' advocates argue that there's more to programming than Scratch, or even than Python; and to avoid children becoming bored with Scratch, they should also learn Logo, Kodu, Hopscotch, Tynker, and so on.

Fluency counts

Somehow though, programming seems different from creating a presentation. If we're serious about students becoming adept at solving problems and expressing their creativity through programming, then they need to develop some sort of mastery of the medium: in short, they need to become fluent (or at least conversant) in one, or two, or three programming languages.

By analogy with learning human languages – while we might be concerned that students get to grips with comparative linguistics and deep structure, we're much more concerned that students learn a language, are able to hold productive conversations in it and, perhaps later, become able to write stories, essays or poems in the language.

There seems to be relatively little research comparing the one-language and many-language approaches to teaching programming: mainly, I suspect, because this question rarely arises in undergraduate CS education. It is simply assumed that introductory programming courses will teach a particular language, although this still allows plenty of argument over the choice of paradigm and language.

While the brightest and best will, I think, be able to transfer the concepts from one language to another, I fear we do a disservice to those who struggle to express their ideas in a first programming language by rapidly introducing them to many more. Changes in syntax, grammar and vocabulary do little more than add to their already high cognitive load. Let's learn the lesson from the mathematics education community, that mastery comes through depth of experience, not superficial acceleration. (HW)

Miles Berry is principal lecturer in computing education at the University of Roehampton. He serves on the boards of Computing At School, the BCS Academy and the Computer Science Teachers Association, and is a Member of the Raspberry Pi Foundation.